A Fast and Accurate Index Structure for Spatiotemporal Trajectories

Somayeh Naderivesal
School of Computing and Information Systems
The University of Melbourne
Parkville, VIC 3010, Australia
naderis@student.unimelb.edu.au

Lars Kulik
School of Computing and Information Systems
The University of Melbourne
Parkville, VIC 3010, Australia
lkulik@unimelb.edu.au

James Bailey
School of Computing and Information Systems
The University of Melbourne
Parkville, VIC 3010, Australia
baileyj@unimelb.edu.au

ABSTRACT

There are many applications that rely on the analysis of historical trajectory data provided by location-aware devices. Traffic management, ridesharing and Pay-As-You-Drive insurance are some example applications. These applications require one to scan a large-scale historical dataset in order to extract similar trajectories to a query trajectory. A naive approach is a one-by-one similarity comparison between the query trajectory and dataset trajectories. However, most existing methods for assessing similarity between pairs of trajectories have quadratic time-complexity, making sequential scan of a large dataset computationally expensive. Existing solutions for this problem mainly focus on reducing the computation time by summarizing the trajectories or by defining upper-bounds (lower-bounds) for the similarity between two trajectories. However, these solutions do not directly address the efficiency challenge. To tackle this challenge of large scale similar trajectory retrieval, we propose an efficient index structure that filters out distant trajectories and outputs an approximate set of similar trajectories. Experimental results show that our novel index structure significantly improves the efficiency of trajectory retrieval, whilst maintaining high accuracy.

CCS CONCEPTS

•General and reference → **Metrics; •Computing methodologies** → *Machine learning algorithms;*

KEYWORDS

Spatiotemporal Trajectory, Indexing, Locality Sensitive Hashing

ACM Reference format:
Somayeh Naderivesal, Lars Kulik, and James Bailey. 2017. A Fast and Accurate Index Structure for Spatiotemporal Trajectories. In *Proceedings of the 14th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Melbourne, VIC, Australia, November 7–10, 2017 (MobiQuitous),* 9 pages.
DOI: 10.1145/3144457.3144497

MobiQuitous, Melbourne, VIC, Australia

DOI: 10.1145/3144457.3144497

1 INTRODUCTION

The widespread use of location-aware devices provides a tremendous opportunity to harness large amounts of stored trajectory data. With the availability of inexpensive storage devices, we are now able to easily store large scale moving object trajectory datasets such as Cabspotting [28] and the Porto taxi trajectory dataset [1].

There are many applications that rely on the analysis of trajectory data. For example, in ridesharing applications such as Uber [3], trajectories are routinely stored to deal with customer complaints and service improvements. For instance, a user might complain about a late arrival or a long path taken by a driver. How should Uber address this complaint? One attractive solution is to compare the user's trajectory with the shortest path that could have been taken. However, for different reasons such as road closures and unforseen road congestion (due to roadworks or traffic incidents), travelling along the shortest path does not always lead to the shortest trip time. Similarly, travelling with the maximum allowed speed is not always feasible. In other words, there are variety of conditions that may impact the movement of a vehicle in a road network at different times during the day.

Our approach is to identify the spatiotemporally similar trajectories and compare them to the user's (query) trajectory. For example, suppose a user complains about a late arrival because of a slow ride, since they believe it was feasible to drive faster. To investigate this complaint, Uber needs to identify trajectories around the same 1) space (within ε distance in each spatial dimension) and 2) time (within δ in the time dimension). Uber is then able to compare the user's trajectory to those trajectories. If there is not any other trajectory within ε distance and δ time, it means that there is not any problem with the ride. Otherwise, Uber must resolve the complaint according to its internal policies. With millions of rides every day, Uber has a large-scale dataset of trajectories. On December 2016, only in New York City, the average number of daily trips was around 240,000 [2]. Finding similar trajectories to the user trajectory in such a huge dataset is computationally expensive if we need to do a sequential scan of the dataset. Taking another example, suppose an Uber customer provides his everyday or preferred trajectory and is looking for a carpooling service. Here, the task is to find the best match for the provided trajectory from existing customer trajectories and recommend them to the customer. In both examples, *given two thresholds ε and δ for the space and time dimensions, we are looking for similar candidate trajectories to a query trajectory from a dataset of trajectories (similar trajectory retrieval).*

The naïve approach for similar trajectory retrieval is to compare the query trajectory against the entire dataset of trajectories using

an appropriate similarity measure. Several similarity measures have been proposed for trajectory comparison [6–8, 26]. Longest Common Subsequence (LCSS) [24, 26] and Dynamic Time Warping (DTW) [6] are two well-studied similarity measures for trajectory data. LCSS and DTW (and also most of the other measures) have quadratic time complexity, which is costly, when we compare the query trajectory across the entire dataset of trajectories. However, for applications that require finding time-sensitive (δ) similar trajectories to a query trajectory within a given spatial distance (ε), DTW is not a viable candidate measure. Although we can use time-constrained DTW [18], it does not consider spatial closeness in trajectory comparisons (ε).

To decrease the computation time of retrieving similar trajectories using LCSS , several methods have been proposed [18, 23, 25]. In [20] and [23], the proposed approaches are not filtering methods and they have applications as refining methods. It means that they do not prune distant trajectories without any comparison between the query trajectory and the entire dataset trajectories. However, in [25], Valchos et al. proposed an index structure that uses multidimensional Minimum Bounding Rectangles (MBRs) to index trajectories which makes the similarity search up to 14 times faster than the sequential scan (Section 5.1). However, in our experiments, we will show that using their indexing approach still incurs high computation time. In summary, we argue that an indexing structure for similar trajectory retrieval must meet the following requirements:

R1 *The indexing method must consider both time and space dimensions*. Indexing spatiotemporal trajectories using either only the spatial or only the temporal dimension is not a versatile solution for different trajectory datasets. For example, indexing only the spatial dimension has low pruning effectiveness when trajectories belong to moving objects that travel the same path every day.

R2 *The indexing method must output a candidate set of similar trajectories that minimises false positives and false negatives*. False positives are trajectories they are incorrectly extracted by the index structure as similar candidate trajectories. False negatives are trajectories that are similar to the query trajectory, but they have been incorrectly recognised as distant trajectories. Our aim is to minimise false positives and false negatives while at the same time achieving high efficiency.

As discussed before, none of the existing approaches meet all the requirements (R1 and R2) for a fast and accurate similar trajectory retrieval. In this paper, we propose a novel method which significantly improves the efficiency of existing methods while it has a high accuracy as well. Our proposed method includes two index structures, one for the spatial dimension and one for the temporal dimension. Having two independent indices for the spatial and the temporal dimensions is beneficial when trajectories are very dense in one of the dimensions. Our spatial index structure is novel and efficient. Using the reverse triangle inequality, we transform two-dimensional spatial points to high-dimensional points. Describing spatial points in higher dimensions enables better differentiation between points. We use Locality Sensitive Hashing [14, 17] to index the high-dimensional points using a small number of hash tables. For temporal indexing, we employ an R-tree [15] to index the time interval of trajectories. Extracting the overlap between a query interval/MBR and the entries (interval/MBR) stored in a R-tree is very fast (O(log n)).

Table 1: Notations used for the problem statement

Notation	Description
DB	A set of n trajectories, $DB = (T_1, T_2, ..., T_n)$
T_i	The i^{th} trajectory of DB, $T_i = (X_1, X_2, ..., X_{\|T_i\|})$
X_j	The j^{th} time-stamped coordinate of T_i, $X_j = (t_j, S_j)$, $S_j = (lat_j, long_j)$
$[t_s^{T_i}, t_e^{T_i}]$	The time interval of trajectory T_i
Q	The query trajectory, $Q = (q_1, q_2, ..., q_{\|Q\|})$
$T(h_i(q))$	The set of trajectories in hash bucket with the value $h_i(p)$ of m^{th} hash table
$\|\|q - r\|\|_2$	The Euclidean distance between points q and r.
S_{Q,T_i}	The approximated similarity value between a query trajectory Q and dataset trajectory T_i
kNN_Q	k most similar trajectories to Q, $kNN_Q \subset DB$
C_Q (C_Q^{exact})	The exact set of similar candidate trajectories, $C_Q \subset DB$
C_Q^{approx}	The approximate set of similar candidate trajectories, $C_Q^{approx} \subset DB$

In summary, our main contributions are as follows:

- We propose a novel method for fast similar trajectory retrieval. It includes two index structures: one spatial and one temporal index to handle different distributions in time and space dimensions (R1).
- We introduce a novel efficient spatial index structure. Using the reverse triangle inequality property and LSH, we ensure a highly efficient similarity search.
- We evaluate the efficiency and effectiveness of our method using two real spatiotemporal datasets. We demonstrate that our proposed method is up to 25 times faster than the indexing method proposed in [24]. We show that our method is also more accurate for similar trajectory retrieval compared to [24] (R2).

In the rest of the paper, we firstly review existing methods for similar trajectory retrieval in Section 2. In Section 3, we describe our novel method for similar trajectory retrieval. Section 4 presents experimental results and we conclude our findings in Section 5.

2 RELATED WORK

Recently, similarity search for trajectory data has been a popular area of research and its background research area, time-series similarity, has also been studied in a number of contexts. Euclidean distance was applied in [4] and [12] to compare time-series with the assumption that time-series have the same number of data points. This assumption limits the application of Euclidean distance measure (and generally Lp-norm measures) [11]. However, the time-complexity of Euclidean distance is $O(n)$ which is an advantage for this measure [10]. Considering "acceleration and decelerations on the rate of sequences" Dynamic Time Warping (DTW) [6, 19] has been proposed to enable time-series to be expanded or compressed to match to each other. In other words, DTW is looking for the maximum similarity

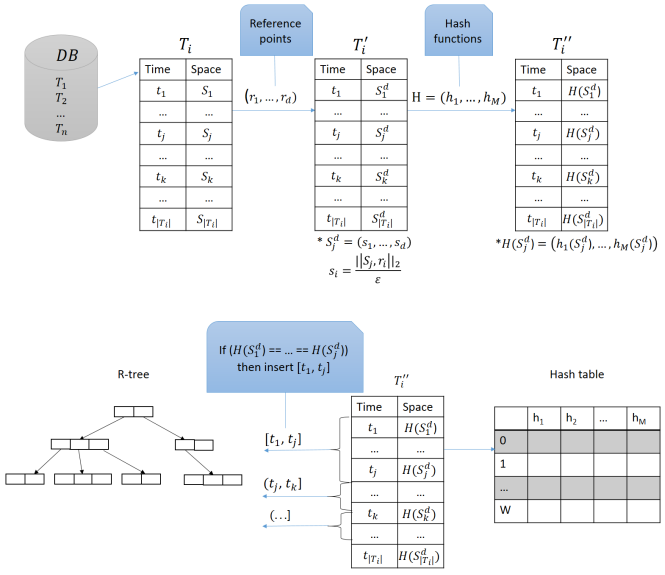


Figure 1: The overall schema of our proposed method for indexing spatiotemporal trajectories

between values and matches values by time warping which makes it a spatial similarity measure. Also, it is very sensitive to noise. In [18], a warping window has been added to DTW for restricting the amount of time shifting in order to increase the accuracy and decrease the time complexity (time-constrained DTW). However, it is still very sensitive to noise. Another measure that also limits the extent of time shifting is LCSS [24, 26]. LCSS has been introduced for trajectory data and utilises the Longest Common Subsequence which is a method for sequence matching. LCSS uses delta δ as a threshold for time shifting and epsilon ε as the maximum difference between spatial dimensions so that they can be considered as a match. Given two trajectories T_1 and T_2, LCSS similarity between them is computed as follows:

$$LCSS_{\delta,\varepsilon}(T_1,T_2) = \begin{cases} 0 & if |T_1| = 0 or |T_2| = 0 \\ 1 + LCSS_{\delta,\varepsilon}(& \\ tail(T_1), tail(T_2)) & if |lat_{|T_i|} - lat_{|T_j|}| < \varepsilon, \\ & |long_{|T_i|} - long_{|T_j|}| < \varepsilon \\ & and\ |t_{|T_i|} - t_{|T_j|}| \leq \delta \\ max(& \\ LCSS_{\delta,\varepsilon}(tail(T_1), T_2), & \\ LCSS_{\delta,\varepsilon}(T_1, tail(T_2))) & otherwise \end{cases} \quad (1)$$

(If $T_1 = (X_1, ..X_{|T_1|})$, $tail(T_1)$ returns $(X_2, ..X_{|T_1|})$)

Two points are counted as a match if their time difference is less than δ and also their spatial distance in every dimension is not greater than ε. The threshold ε makes LCSS robust to noise.

Edit distance with Real Penalty (EDR) [8], Edit Distance on Real sequence (ERP) [7] and Edit Distance with Projections (EDwP) [21] have been proposed to measure spatial similarity between trajectories. Similar to original DTW and LCSS, they have quadratic time-complexity which makes them computationally expensive.

DISSIM [13] has been introduced for spatiotemporal similarity measurement with linear time-complexity. However, if we are required to scan a big dataset, a linear time-complexity for each comparison is still computationally expensive. Also, it does not consider a minimum threshold (ε) for the spatial similarity of trajectories.

For both time-constrained DTW and DISSIM, all the trajectories that have a temporal similarity with the query trajectory are a similar candidate trajectory. Also, both of them are very sensitive to noise. On the other hand, LCSS uses both of the thresholds for similarity comparison and is less sensitive to noise. Whilst this makes it more attractive for our context, as discussed before it has quadratic time-complexity which is a limitation for similarity search in large spatiotemporal datasets.

In [20], Morse et al. proposed a dynamic programming approach to make the similarity search faster for LCSS. However, as they explained, it is an approach that can be used as the refinement step, after filtering the distant trajectories using an index structure. Sherkat et al. proposed [23] two methods for fast similarity search using LCSS. In the first method, they summarise the trajectories and in the second method, it defines lower-bounds and upper-bounds to makes the similarity search faster. Both approaches need a sequential scan of the dataset which is expensive for large trajectory datasets.

In [25], Valchos et al. proposed an index structure that uses MBRs to index spatiotemporal trajectories. In the experiments, we will show that it makes the similarity search 14 times faster than the sequential scan (Section 5.1), however, it still incurs high computation time for large trajectory datasets.

3 PROPOSED METHOD

In this section, we will describe our novel similar trajectory retrieval method.

In many spatiotemporal data analysis tasks, we need to compare a query trajectory (Q) against trajectories in a historical trajectory dataset DB, in order to identify a set of similar candidate trajectories C_Q to the query trajectory Q. However, as discussed before, the sequential scan of DB to compute the exact set of candidates is computationally expensive. In this paper, we propose an extremely fast method that prunes a tremendous number of distant trajectories, before similarity comparison. The remaining trajectories form an *approximate* set of similar candidate trajectories C_Q^{approx}.

Our proposed method indexes spatiotemporal trajectories to a spatial and a temporal index structure. To build the spatial index, we transform 2-dimensional spatial points to high-dimensional points and then we use locality sensitive hashing (LSH) to index the high-dimensional points using a small number of hash tables (Figure 1). To index the temporal dimension, we use a R-tree. We partition the time interval of each dataset trajectory based on its spatial hash values. If two points have the same hash values, their time values are in the same partition. Then, we insert those partitions into a R-tree (Figure 1). To find the approximate set of similar candidate trajectories to a given a query trajectory Q, our proposed method searches 1) the R-tree for trajectories that have a temporal overlap

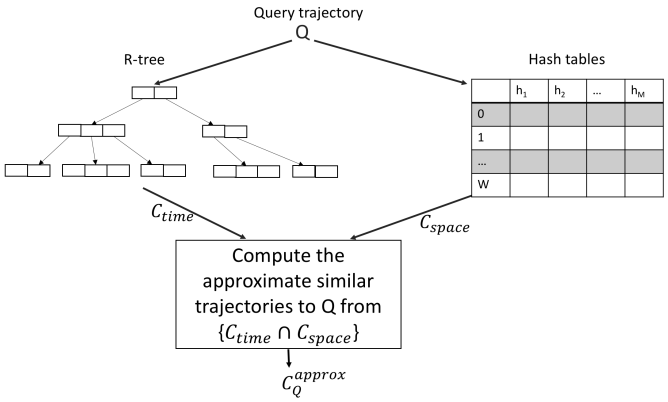


Figure 2: Similar candidate trajectory retrieval using our proposed index structure

with Q and 2) the hash tables for trajectories that are spatially close to Q (Figure 2).

To develop a spatial index structure, we use the following statement which is a consequence of the reverse triangle inequality: *if the Lp-norm distance of two points from each other is less than or equal to ε, then the difference between their distances from a reference point is also less than or equal to ε.*

For example, in Figure 3(a), q is a query point, r_1 is a reference point and also ε is equal to 1. The (Euclidean) distance between points q and S_2 is equal to 1 and the difference between their distances from r_1 is $2.24 - 2 \leq 1$.

Also, the inverse statement says that if the difference between the distance of two points from a reference point is greater than ε, then the distance between them is also greater than ε. For example, in Figure 3(a), the difference between the distance of q and S_3 from r_1 is $3.5 - 2 = 1.5$ and the distance between them is 1.6 which is also greater than 1. Looking for similar candidate points to a query point, the inverse statement helps us to prune some of the points (such as S_3) that are far from the query point. We index the points to a hash table, based on their distances from a reference point. Then, for every query point, we compute its distance from the reference point and find those points that have the same distance to the reference point.

However, the converse statement is not always true: if two points have the distance less than or equal to ε from a reference point, their distance from each other is *not necessarily* less than or equal to ε. For example, point S_1 will be incorrectly extracted as a similar point to q (q and S_1 both have the distance of 2 from the reference point r_1), while the distance between them is 4. To address this problem, we use multiple reference points. If two points are close to each other, then they have similar distances from multiple reference points. This property helps us to decrease the number of false positives and false negatives. For example, in Figure 3(b), if we add another reference point r_2, we are able to prune S_1 from similar candidate points of q. This is because the distance between q and r_2 is 1.41, while the distance between S_1 and r_2 is 3.16 ($3.16 - 1.41 > 1$).

Using multiple reference points, we describe a spatial point in high dimensions which provides us a more precise definition of each

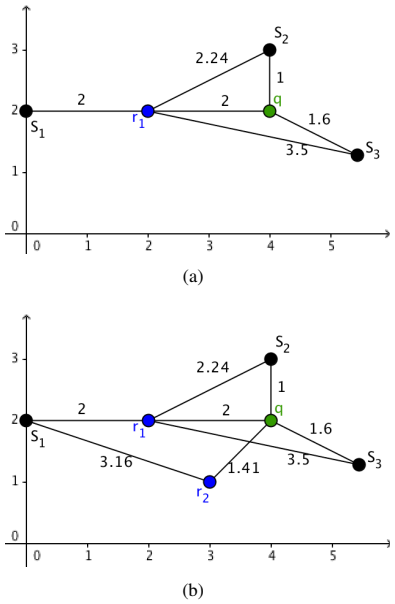


Figure 3: (a) Using reference point r_1, points S_1 and S_2 are similar candidate points to q and point S_3 is pruned. (b) Adding reference point r_2, both point S_1 and S_3 are pruned

point. However, as the size of the dataset increases, the number of reference points should be increased which incurs more memory to store their corresponding hash tables. To address this problem, we use Locality Sensitive Hashing (LSH). LSH shows a high performance in approximate nearest neighbor search in high-dimensional data [5, 9, 14, 22, 27]. To solve the memory problem, we use LSH to index the high-dimensional points to a few number of hash tables. In other words, first, we transform 2-dimensional spatial points into d dimensions using d reference points and then we use LSH to index the d-dimensional points to M hash tables where $M << d$.

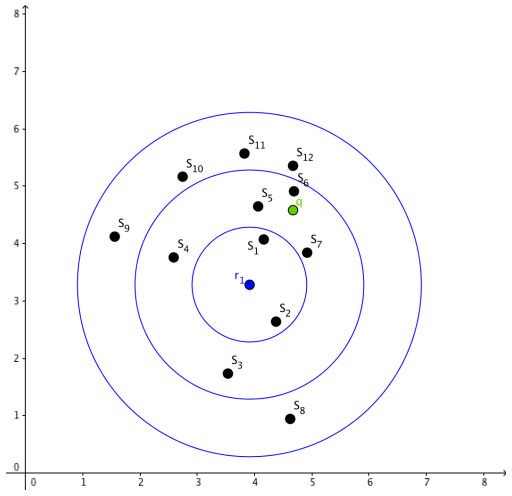
However, for the temporal indexing, we use a R-tree [] to find all the trajectories that have a temporal overlap with the query trajectory. To decrease the number of false positives, we partition the time interval of trajectories based on their hash values and then insert the partitions into the R-tree (Figure 1). The intersection of the spatial and the temporal similar trajectories results in the candidate set of spatiotemporal similar trajectories to the query trajectory Q (Figure 2).

In the rest of this section, first, we describe the spatial index structures in section 3.1. In section 3.2, we describe the temporal index structure. Then, we explain the approximate similar trajectory retrieval using the proposed index structures in section 3.3.

3.1 Spatial index structure

Most of the existing similarity (distance) measures use Lp-norm in metric space to compute the spatial similarity (distance) between trajectories. The following lemma shows a property of Lp-norm which we use it for the pruning of distant points.

Lemma 1: Let S_1, S_2 and S_3 be three points in metric space. If the Lp-norm distance ($p \geq 1$) between S_1 and S_2 is less than

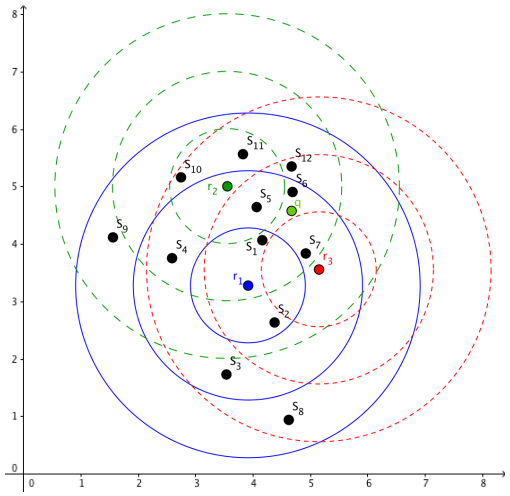


(a)

Bucket	Points (r_1)
[0,1]	{S_1, S_2}
(1,2]	{S_3, S_4, S_5, S_6, S_7} q
(2,3]	{S_8, S_9, S_{10}, S_{11}, S_{12}}
(3,∞)	{}

(b)

Figure 4: Using a single reference point r_1, we extract $S_3, ..., S_7$ as similar candidate trajectories to q, while S_3 and S_4 are false positives. Also, S_1 and S_{12} are false negatives ($\varepsilon = 1$)



(a)

Bucket	Points (r_1)	Points (r_2)	Points (r_3)
[0,1]	{S_1, S_2}	{S_5, S_{10}, S_{11}}	{S_7}
(1,2]	{S_3, S_4, S_5, S_6, S_7} q	{S_1, S_4, S_6, S_7, S_{12}} q	{S_1, S_2, S_5, S_6, S_{12}} q
(2,3]	{S_8, S_9, S_{10}, S_{11}, S_{12}}	{S_2, S_9}	{S_3, S_4, S_8, S_{10}, S_{11}}
(3,∞)	{}	{S_3}	{S_9}

(b)

Figure 5: Adding two reference points r_2 and r_3 to Figure 4(a), helps to decrease the number of false positive and false negatives

or equal to ε ($||S_1 - S_2||_p \leq \varepsilon$), then the difference between their Lp-norm distances from a third point S_3 is less than or equal ε ($|||S_1 - S_3||_p - ||S_2 - S_3||_p| \leq \varepsilon$).

PROOF. Proof is immediate:

$$
\begin{aligned}
&|||S_1 - S_3||_p - ||S_2 - S_3||_p| \\
&= |||S_1 - S_3||_p - ||-S_2 + S_3||_p| \ \textit{reverse triangular inequality} \\
&\leq ||S_1 - S_3 - S_2 + S_3||_p = ||S_1 - S_2||_p \leq \varepsilon
\end{aligned}
$$

□

Using this property, if we index spatial points based on their distances from a reference point, we are able to prune those points that do not have the same distance as the query point from a reference point, as distant points. Figure 4(a) illustrates points $S_1, ..., S_{12}$, a reference point r_1 and a query point q. Figure 4(b) is a hash table which indexes the points S_1 to S_{12} based on their distances from r_1. To find the points within the distance of 1 ($\varepsilon = 1$) from q, we are able to find a candidate set of similar points (S_3 to S_7) from its respective hash bucket ((1,2]) and prune other points as distant points.

However, S_3 and S_4 are not within the distance of 1 from q, while they are in the same hash bucket (false positives). Also, the distance between q and S_1/S_{12} is less than 1, while they are in different hash buckets from q (false negatives). To decrease the number of false positives and false negatives, we can use multiple reference points. In Figure 5, we added two new reference points r_1 and r_2 to the points in Figure 4(a). If we consider the points that are in the same bucket as q *at least for two reference points*, the candidate set excludes S_3 and includes both S_1 and S_{12}. In this way, there is no false negative (no similar point is missing) and there is just one false positive (S_4).

Definition 1: Given d reference points $r_1, ..., r_d \in \mathbb{R}^2$, function $f : \mathbb{R}^2 \to \mathbb{R}^d$, transforms S to S^d as the following:

$$S^d = f(S) = (s_1, ..., s_d),\ s_i = \lfloor \frac{||S,\ r_i||_2}{\varepsilon} \rfloor \tag{2}$$

As discussed before, if we increase the number of reference points, the accuracy of nearest neighbor search increases. However, the drawback is that more memory space is required to store the hash tables. To address this problems, we use Locality Sensitive Hashing (LSH). LSH has been widely used for finding approximate nearest neighbors in high-dimensional data. LSH indexes high-dimensional points into hash buckets so that close points are mapped to the same buckets with high probability.

Definition 2: A family of hash functions $H = \{h : U_1 \to U_2\}$ is called $(R_1, R_2, P_1, P_2) - sensitive$ if the following conditions are satisfied for any two points y_1 and y_2 from U_1:

(1) $If\ dist(y_1, y_2)\ \leq\ R_1\ then\ pr_H(h(y_1) = h(y_2)) \geq P_1$

(2) $If\ dist(y_1, y_2)\ \geq\ R_2\ then\ pr_H(h(y_1) = h(y_2)) \leq P_2$

Definition 2: Let $S^d \in \mathbb{R}^d$ ($S^d = (s_1, ..., s_d)$ and $s_i \in \mathbb{R}$) and $A = (a_1, ..., a_d)$ while $a_1, ..., a_d$ are drown i.i.d from distribution Z. The

distribution Z is p-stable, if there exists $z > 0$ such that $\sum_i s_i a_i$ has the same distribution as the variable $(\sum_i |s_i|^p)^{1/p} z$ where z is a random variable from distribution Z.

Using the fact that a Gaussian distribution $N(0,1)$ is 2-stable, Datar et al. [9] proposed the following hash function for LSH:

$$h_i(S^d) = \lfloor \frac{A_i . S^d + B_i}{W} \rfloor \quad (3)$$

Using LSH, the objective is to approximate nearest neighbors of a query point q using M hash functions, while $M << d$:

$$H(S^d) = (h_1(S^d), h_2(S^d), ..., h_M(S^d)) \quad (4)$$

where A_i consists of i.i.d. entries $a_1, ..., a_d$ from Gaussian distribution $N(0,1)$ and B_i consists of $b_1, ..., b_d$ which are drawn from uniform distribution $U[0, W)$ (W controls the number of hash buckets).

The index structure for the spatial dimension consists of M hash tables, one per hash function. As described before, to build the index, first, we transform 2-dimensional spatial data to d dimensions ($d > 2$) and then we use LSH to index d-dimensional data to M hash tables ($M << d$). Consider that the value of d is highly dependent on the number of the points and also their distribution on the space.

In summary, given d reference points $\{r_1 ... r_d\}$ and M hash functions $\{h_1, ..., h_M\}$, we index spatial points of trajectory $T_i \in DB$ as the following:

(1) For every spatial point $S_j \in T_i$:

  (a) Compute the d-dimensional vector S_j^d using (2)

  (b) Using (3) and (4), compute $H(S_j^d)$

  (c) For every $h_m(S_j^d) \in H(S_j^d)$ ($m = 1..M$), add i (the unique identifier of the trajectory) to bucket $b_l = h_m(S_j^d)$ in m^{th} hash table, if $i \notin b_l$

In this way, bucket b_l of a hash table h_m, contains the index of all trajectories that have at least one point in this bucket.

In order to prevent from having big hash tables, we set a limit for the number of hash buckets. For example, in Figure 4(b) and Figure 5(b), the number of hash buckets are limited to four. However, taking this limit, we must be sure that there are some reference points from the areas that have a high density in terms of the number of points. Otherwise, the number of false positives will be increased.

However, there are different methods to choose reference points. In this paper, we use two approaches: density-based in which we select the reference points using density-based clustering (k-means) when the number of clusters is d, and random selection in which we select d reference points i.i.d from the bounding box that contains the entire set of trajectories.

3.2 Temporal index structure

To index the time dimension, we use a 1-dimensional R-tree. R-tree [15] is a tree data structure proposed for indexing spatial data. It has been widely used for different applications including geographical data, video/image data and chronological data such as time intervals.

The naïve approach to index the time dimension of trajectories is indexing the entire time interval of trajectories $[t_s^{T_i}, t_e^{T_i}]$ as a unique entry, in the R-tree. Then, to find similar trajectories to a query trajectory Q in time dimension $[t_s^Q, t_e^Q]$, we search the R-tree for all the trajectories that have an overlap with $[t_s^Q, t_e^Q]$.

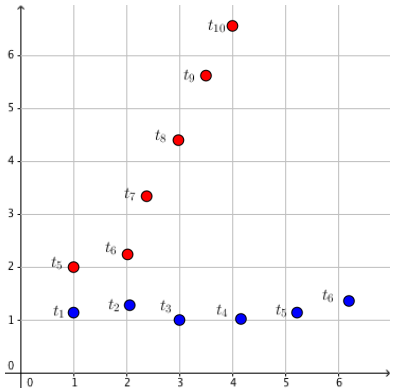


Figure 6: Two trajectories T_1 and T_2 which have separate spatial and temporal similarities (they are not spatiotemporally similar)

However, indexing the spatial and the temporal dimensions completely independent, might result higher rate of false candidates. The reason is that the spatial and temporal overlaps of a candidate trajectory with a query trajectory might be independent. Figure 6 illustrate two sample trajectories that have separate overlaps in time and space dimension. For the first two points of the trajectories, they have spatial similarity (given $\varepsilon = 1$), while their temporal similarity is the first two points of T_1 and last two points of T_2. Then, they are not spatiotemporally similar.

To address this problem, we use the hash values of spatial points to partition the time interval to multiple intervals. We partition the time interval of T_i ($[t_s^{T_i}, t_e^{T_i}]$), when the hash values of a spatial point $H(S_j^d)$ is different from $H(S_{j-1}^d)$. To build both spatial and temporal indices, for every trajectory $T_i \in DB$, we perform the following steps:

(1) $t_s = t_1$

(2) For every $X_j = (t_j, S_j) \in T_i$

  (a) Compute the d-dimensional vector S_j^d using (2).

  (b) Using (3) and (4), compute $H(S_j^d)$

  (c) For every $h_m(S_j^d) \in H(S_j^d)$ ($m = 1..M$), add i (the unique identifier of the trajectory) to bucket $b_l = h_m(S_j^d)$ in m^{th} hash table, if $i \notin b_l$

  (d) If $H(S_j^d) <> H(S_{j-1}^d)$ then insert the interval $[t_s, t_{j-1}]$ to R-tree and set t_s to t_{j-1}

(3) Insert $[t_s, t_{|T_i|}]$ to the R-tree

3.3 Similar trajectory retrieval

To retrieve similar candidate trajectories to the query trajectory Q, we use the spatial and the temporal index structures to extract similar trajectories in each dimension. Then, the intersection between these two sets is the final set of candidates (Figure 2).

The ordering of steps for approximate similar trajectory retrieval is as the following:

Table 2: The datasets used in the experiments

Dataset	Number of trajectories	$(Avg. \pm Std.)$ of sampled points in trajectories
Cabspotting dataset	32000	(257 ± 180)
Porto taxi dataset	$1,700,000$	(411 ± 222)

(1) For every $X_j = (t_j, S_j) \in Q$:

(a) Compute the d-dimensional vector S_j^d using (1)

(b) Compute $H(S_j^d) = (h_1(S_j^d), ..., h_M(S_j^d))$

(c) Compute $C_{space} = \{T(h_1(S_j^d)) \cup ... \cup T(h_M(S_j^d))\}$ where $T(h_m(S_j^d))$ returns the identifier of all trajectories that are in the bucket $b_l = h_m(S_j^d)$ of the m^{th} hash function

(d) Search R-tree for trajectories that have a time interval which includes t_j and include them in set C_{time} (if we have threshold δ, then we search $[t_{S_j} - \delta, t_{S_j} + \delta)$

(e) For every $i \in C_{space}$, add 1 to the similarity value of Q and T_i ($S_{Q,T_i} = S_{Q,T_i} + 1$), if T_i is similar to Q in time dimension as well ($i \in C_{time}$)

(2) $C_Q^{approx} = \{T_i \in DB, S_{Q,T_i} <> 0\}$

Then, C_Q^{approx} contains the trajectories that their approximate similarity value to Q (S_{Q,T_i}) is non-zero. In the experiments, we will do some further analysis on the approximate value. We compute the correlation between the approximated and the exact value of similarity.

4 EXPERIMENTS

In this section, we present an empirical study on our proposed method using two real datasets. We verify the effectiveness and efficiency of similar trajectory retrieval using 1) our method and 2) the proposed method by Valchos et al. [25].

Datasets: We use two traffic datasets, Cabspotting and Porto taxi trajectory dataset [1]. Cabspotting contains one-week trajectories of 536 taxis in San Francisco and has about 15 million of points. In Cabspotting dataset, all the data for a taxi is in one stream. We partition the trajectories of these datasets when the moving objects are stationary for more than 15 minutes which results a total number of 32 thousands trajectories. However, Porto taxi trajectory dataset contains trajectories of 442 taxis over a year in Porto, Portugal. The number of trajectories in Porto dataset is 1.7 million trajectories. Table 2 illustrates a summary of the datasets including the number of sampled points in trajectories and the average and the standard deviation of number of sampled points in trajectories.

Comparison criteria: For all the applications that we discussed in Section 1, we need to define two thresholds for both spatial and temporal similarity, i.e. we are looking for trajectories that are within the distance of ε from the query trajectory, while we are flexible in time dimension by δ. Since LCSS defines the similarity between trajectories on both dimensions, we use LCSS to build the ground truth of the experiments. Ground truth for a query trajectory Q is the set of similar trajectories C_Q^{exact} (C_Q) that are extracted in a sequential scan, using LCSS measure. On the other hand, C_Q^{approx} is the set of similar candidate trajectories extracted by our/Valchos et al. proposed method. We compute the effectiveness of both methods by computing the following measures for every experiment.

Table 3: The accuracy of similar trajectory retrieval for Porto taxi dataset ($\delta = 1 minute$)

$\delta = 1$ *Minute*								
Measure:	Correlation		Recall		Precision		F-Score	
Method:	Ours	[25]	Ours	[25]	Ours	[25]	Ours	[25]
$\varepsilon = 1km$	0.55	0.67	0.95	0.72	0.31	0.48	0.47	0.58
$\varepsilon = 3km$	0.58	0.61	0.94	0.79	0.63	0.71	0.75	0.75
$\varepsilon = 5km$	0.67	0.54	0.92	0.72	0.75	0.76	0.82	0.74
$\varepsilon = 7km$	0.67	0.45	0.92	0.64	0.79	0.77	0.85	0.70
$\varepsilon = 10km$	0.68	0.36	0.91	0.52	0.8	0.78	0.85	0.62

Table 4: The accuracy of similar trajectory retrieval for Porto taxi dataset ($\delta = 5 minutes$)

$\delta = 5$ *Minutes*								
Measure:	Correlation		Recall		Precision		F-Score	
Method:	Ours	[25]	Ours	[25]	Ours	[25]	Ours	[25]
$\varepsilon = 1km$	0.37	0.58	0.95	0.83	0.40	0.60	0.56	0.70
$\varepsilon = 3km$	0.58	0.61	0.94	0.84	0.71	0.79	0.70	0.80
$\varepsilon = 5km$	0.63	0.59	0.93	0.78	0.81	0.83	0.86	0.80
$\varepsilon = 7km$	0.63	0.55	0.92	0.70	0.84	0.83	0.88	0.76
$\varepsilon = 10km$	0.65	0.52	0.92	0.59	0.85	0.83	0.88	0.68

Table 5: The accuracy of similar trajectory retrieval for Porto taxi dataset ($\delta = 10 minutes$)

$\delta = 10$ *Minutes*								
Measure:	Correlation		Recall		Precision		F-Score	
Method:	Ours	[25]	Ours	[25]	Ours	[25]	Ours	[25]
$\varepsilon = 1km$	0.33	0.54	0.98	0.82	0.49	0.66	0.64	0.73
$\varepsilon = 3km$	0.53	0.62	0.95	0.87	0.83	0.83	0.90	0.84
$\varepsilon = 5km$	0.57	0.62	0.92	0.82	0.91	0.87	0.91	0.84
$\varepsilon = 7km$	0.57	0.61	0.92	0.74	0.93	0.88	0.92	0.80
$\varepsilon = 10km$	0.58	0.56	0.92	0.63	0.94	0.92	0.92	0.73

$$recall = \frac{|C_Q^{approx} \cap C_Q^{exact}|}{|C_Q^{exact}|} \tag{5}$$

$$precision = \frac{|C_Q^{approx} \cap C_Q^{exact}|}{|C_Q^{approx}|} \tag{6}$$

$$F - score = 2. \frac{precision.recall}{precision + recall} \tag{7}$$

$$\rho_{C_Q^{exact}, C_Q^{approx}} = \frac{cov(C_Q^{exact}, C_Q^{approx})}{\sigma_{C_Q^{exact}} \sigma_{C_Q^{approx}}} \tag{8}$$

$\rho_{C_Q^{exact}, C_Q^{approx}}$ is the Pearson correlation coefficient between C_Q^{exact} and C_Q^{approx}.

Platform: All measures are implemented in C programming language and run on a computer with an Intel Core i5 CPU (2.7 GHz) and 8 GB memory.

Parameters: There are five different parameters in the experiments. Based on the applications that we discussed, we set the value of δ to 1, 5 and 10 minutes and ε to 1, 3, 5, 7 and 10 kilometres. The number of reference points d are selected based on the spatial extent of each dataset. We examined different values of d for each dataset, however we show the results for the minimum value which has the best outcome (200 for both datasets). Also, the number of hash functions M and the size of hash buckets W are constant and are equal to 10 and 1000 respectively, in all the experiments. *Increasing the number of reference points and hash functions will increase the accuracy of the our method, however, it will increase the required memory space for the index structure.*

4.1 Indexing Accuracy

In this section, we investigate the accuracy of our proposed method and also the method in [25]. As discussed before, we compare the results of these two methods with the ground truth that we explained before. For every query trajectory, we compute the approximate similar candidate trajectories C_Q^{approx} using our method and method in [25]. Also, we compute the exact set of similar candidate trajectories C_Q^{exact} to the query trajectory using LCSS. Then, we compute the Pearson correlation coefficient, recall, precision and F-Score between C_Q^{exact} and C_Q^{approx}. Tables 3 to 8 shows the result for both datasets.

Tables 3 to 5 illustrate the results for Porto taxi dataset. As shown in Table 1, increasing ε improves the correlation between our approximated similarity values and the exact similarity values. As we discussed before, d (the number of reference points) and W (the size of hash tables) are constant for all the experiments. The increase in the correlation illustrates that if we increase one or two of these parameters for smaller values of ε (0.03 and 0.05), the correlation value would be higher. Our further experiments has confirmed that, however, we do not want to increase the memory size of our index structure. However, for the other method, the correlation is decreasing. This is because, the method in [25] overestimates the similarity between trajectories and increasing ε reinforces the overestimation. For the same reason, its recall is decreasing and its precision is increasing.

Comparing results in Tables 3, 4 and 5 to each other, when we increase δ in Porto taxi dataset experiments, the increase in F-Score and decrease in correlation is more sensible for both methods. In other words, the accuracy is sensitive to both δ and ε parameters in this dataset. It illustrates that the trajectories in Porto taxi dataset are similar to what we showed in Figure 6. It means that their time and space overlaps are separated for small values of δ and ε. However, when we increase the value of one or both parameters, they become more similar in the new extent.

Table 6 to 8 show the results for Cabspotting dataset. As shown in Table 6, when we increase ε, it does not have an important impact on recall (while the impact is more for Porto taxi dataset). The reason is that the Cabspotting trajectories are not as dense as Porto taxi dataset trajectories in space dimension (we only have 32 thousands trajectories in Cabspotting, while the number of trajectories in Porto dataset is 1.7 million). However, when ε is constant and we increase δ (comparing Tables 6, 7 and 8 to each other), it improves the accuracy. This is because trajectories in Cabspotting dataset are for one week time and they are very dense in the time dimension.

However, considering two different datasets with different characteristics, our proposed method has a higher accuracy in comparison to [25]. In the next section, we will show that in addition to the higher accuracy, we guarantee a lower computation time as well.

Table 6: The accuracy of similar trajectory retrieval for Cabspotting dataset (δ = 1 *Minute*)

δ = 1 *Minute*								
Measure:	Correlation		Recall		Precision		F-Score	
Method:	Ours	[25]	Ours	[25]	Ours	[25]	Ours	[25]
$\varepsilon = 1km$	0.26	0.37	0.99	0.95	0.54	0.56	0.70	0.70
$\varepsilon = 3km$	0.48	0.55	0.99	0.96	0.72	0.67	0.82	0.80
$\varepsilon = 5km$	0.53	0.59	0.98	0.96	0.75	0.69	0.85	0.80
$\varepsilon = 7km$	0.56	0.61	0.98	0.96	0.76	0.70	0.86	0.81
$\varepsilon = 10km$	0.58	0.62	0.98	0.96	0.77	0.71	0.86	0.81

4.2 Indexing Efficiency

In this section, we will verify the efficiency (time and space) of the methods that we discussed before. Table 9, shows the average computation time for finding similar candidate trajectories to a query trajectory using our method, the indexing method in [25] and the grand truth computed by LCSS. Since the computation time for different values of δ and ε are very similar, we only show an average value for each method.

Although the number of trajectories in Cabspotting are 53 times less than the trajectories in Porto taxi dataset (Table 2), the similar trajectory retrieval from Cabspotting is about 13 times slower in

Table 7: The accuracy of similar trajectory retrieval for Cabspotting dataset (δ = 5 *Minutes*)

δ = 5 *Minutes*								
Measure:	Correlation		Recall		Precision		F-Score	
Method:	Ours	[25]	Ours	[25]	Ours	[25]	Ours	[25]
$\varepsilon = 1km$	0.32	0.41	0.99	0.95	0.62	0.64	0.76	0.76
$\varepsilon = 3km$	0.50	0.56	0.99	0.96	0.75	0.68	0.85	0.80
$\varepsilon = 5km$	0.55	0.59	0.98	0.96	0.78	0.70	0.87	0.80
$\varepsilon = 7km$	0.56	0.60	0.98	0.96	0.79	0.71	0.87	0.81
$\varepsilon = 10km$	0.60	0.61	0.95	0.96	0.82	0.72	0.88	0.82

Table 8: The accuracy of similar trajectory retrieval for Cabspotting dataset (δ = 10*minutes*)

δ = 10 *Minutes*								
Measure:	Correlation		Recall		Precision		F-Score	
Method:	Ours	[25]	Ours	[25]	Ours	[25]	Ours	[25]
$\varepsilon = 1km$	0.36	0.41	0.99	0.96	0.61	0.63	0.76	0.76
$\varepsilon = 3km$	0.53	0.54	0.99	0.97	0.73	0.72	0.84	0.82
$\varepsilon = 5km$	0.57	0.58	0.99	0.97	0.76	0.72	0.86	0.82
$\varepsilon = 7km$	0.59	0.59	0.99	0.97	0.77	0.73	0.87	0.83
$\varepsilon = 10km$	0.60	0.60	0.99	0.97	0.78	0.73	0.87	0.83

Table 9: The average computation time of similar trajectory retrieval (millisecond)

Dataset	Ours	[25]	LCSS
Cabspotting dataset	400	10000	90000
Porto taxi dataset	50	500	7000

comparison to Porto taxi dataset using LCSS (ground truth). The reason is that the Cabspotting dataset is very dense in the time dimension in comparison to Porto taxi dataset, about 536 vehicles in one weak vs. 442 vehicles in one year (because of that, the number of similar candidate trajectories to a given trajectory for Cabspotting is about 10 times more than the candidate numbers for Porto taxi dataset).

In summary, our method outperforms the competing method in [25] for both datasets.

Moreover, we measured the memory space used in both index structures. Cabspotting dataset does not need a considerable amount of memory space for similar candidate trajectory computation. However, since Porto taxi dataset is a huge dataset, it needs more memory space for both index structures. Using the method in [25] for Porto taxi dataset, it needs about 500Mb of space, while, our approach needs 1.2Gb for in-memory similar trajectory retrieval. However, this amount of memory space is not an issue for the existing computers in contrast to the time parameter which is critical in most of the computations.

5 CONCLUSION

In this paper, we proposed an indexing method for spatiotemporal trajectories for 1) fast and 2) accurate similar trajectory retrieval. Our proposed method uses two index structures: the R-tree for the temporal dimension and a novel index structure for the spatial dimension. Our novel spatial index structure, first transforms the 2-dimensional spatial points to high-dimensional spatial points. Using Locality Sensitive Hashing, it indexes the high-dimensional data to a limited number of hash tables. The experimental results on two real datasets demonstrate the our method is more accurate than the existing method in [25]. Also, our method is significantly faster than a sequential scan of the dataset for similar trajectory retrieval, since it filters out most of the dataset trajectories that are not similar to the query trajectory considering a space and time threshold. In addition, it is up to 25 times faster than the competing method in [25].

REFERENCES

[1] 2017. Porto taxi trajectory dataset. (July 2017). http://www.geolink.pt/ecmlpkdd2015-challenge

[2] 2017. Taxi, Uber, and Lyft Usage in New York City. (July 2017). http://toddwschneider.com/posts/taxi-uber-lyft-usage-new-york-city/

[3] 2017. Uber. (July 2017). https://www.uber.com/

[4] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. *Efficient similarity search in sequence databases*. Springer.

[5] Alexandr Andoni, Piotr Indyk, Huy L Nguy?n, and Ilya Razenshteyn. 2014. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1018–1028.

[6] Donald J Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *KDD workshop*, Vol. 10. Seattle, WA, 359–370.

[7] Lei Chen and Raymond Ng. 2004. On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 792–803.

[8] Lei Chen, M Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 491–502.

[9] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. ACM, 253–262.

[10] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. of the VLDB Endowment* 1, 2 (2008), 1542–1552.

[11] Philippe Esling and Carlos Agon. 2012. Time-series data mining. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 12.

[12] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. In *Proc. of SIGMOD*. ACM.

[13] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. 2007. Index-based most similar trajectory search. In *Proc. of ICDE*. IEEE, 816–825.

[14] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.

[15] Antonin Guttman. 1984. *R-trees: A dynamic index structure for spatial searching*. Vol. 14. ACM.

[16] John A Hartigan and JA Hartigan. 1975. *Clustering algorithms*. Vol. 209. Wiley New York.

[17] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

[18] Eamonn Keogh and Chotirat Ann Ratanamahatana. 2005. Exact indexing of dynamic time warping. *Knowledge and information systems* 7, 3 (2005), 358–386.

[19] Eamonn J Keogh and Michael J Pazzani. 1998. An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback.. In *KDD*, Vol. 98. 239–243.

[20] Michael D Morse and Jignesh M Patel. 2007. An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 569–580.

[21] Sayan Ranu, P Deepak, Aditya D Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 999–1010.

[22] Venu Satuluri and Srinivasan Parthasarathy. 2012. Bayesian locality sensitive hashing for fast similarity search. *Proceedings of the VLDB Endowment* 5, 5 (2012), 430–441.

[23] Reza Sherkat and Davood Rafiei. 2008. On efficiently searching trajectories and archival data for historical similarities. *Proceedings of the VLDB Endowment* 1, 1 (2008), 896–908.

[24] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. 2003. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 216–225.

[25] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. 2006. Indexing multidimensional time-series. *The VLDB JournalThe International Journal on Very Large Data Bases* 15, 1 (2006), 1–20.

[26] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE, 673–684.

[27] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. 2014. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927* (2014).

[28] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*. ACM, 99–108.